ARMY RESEARCH LABORATORY

The True Limitations of Shared Memory Programming

by D. M. Pressel, M. Behr,
and S. Thompson

ARL-TR-2147 January 2000

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Abstract

Shared memory parallel computers have the reputation for being the easiest type of parallel computers to program. At the same time, they are frequently regarded as being the least scalable type of parallel computer. In particular, shared memory parallel computers are frequently programmed using a form of *loop-level parallelism* (usually based on some combination of compiler directives and automatic parallelization). However, in discussing this form of parallelism, the experts in the field routinely say that it will not scale past 4–16 processors (the number varies among experts). This report investigates what the true limitations are to this type of parallel programming. The discussions are largely based on the experiences that the authors had in porting the Implicit Computational Fluid Dynamics Code (F3D) to numerous shared memory systems from SGI, Cray, and Convex.

Table of Contents

INTENTIONALLY LEFT BLANK.

1. Introduction

Shared memory parallel computers are generally regarded as being the easiest type of parallel computer to program [1]. This statement primarily applies to two types of systems, both of which are examples of shared memory **SMPs**:

(1) Multiprocessor vector computers that in general lack caches (e.g., the Cray C90).

(2) Multiprocessor microprocessor-based systems (either **RISC** or **CISC**) with coherent memory systems that include one or more layers of cache.

Additionally, a few companies have produced systems with *globally addressable* memory, which they referred to as shared memory. The problem with this is that some of these systems are not cache coherent. Two of the more notable examples of this approach are the Cray T3D and the Cray T3E. Since the Cray T3D and T3E support the CRAFT programming model, which uses compiler directives to implement *loop-level parallelism*, the combination of the Cray T3D with the CRAFT model is also discussed.

When discussing *high performance computing* with people in industry and the government, the general goal is to obtain high levels of performance. On the other hand, when one of us has discussed issues related to *high performance computing* and *parallel processing* with graduate students, professors, and researchers in academia, a second point of view has frequently surfaced. In that point of view, large memory uniprocessors and shared memory systems cost too much. Therefore, one of their main goals of using parallel processors was to avoid the performance

Note: This work was made possible through a grant of computer time by the Department of Defense (DOD) High Performance Computing Modernization Program. Additionally, it was funded as part of the Common High Performance Computing Software Support Initiative administered by the DOD High Performance Computing Modernization Program.

Note: Definitions for boldface text can be found in the Glossary.

degradation of using *out-of-core* solvers at a fraction of the cost for traditional large memory systems. This point of view was expressed both by people involved in creating scalable libraries for distributed memory architectures using message-passing code and by a graduate student working in the field of *software distributed shared memory*.

In this report, we are principally concerned with the achievement of high levels of performance. Based on experiences at the U.S. Army Research Laboratory (ARL) and elsewhere, it is our belief that the use of software distributed shared memory (as part of the parallelization strategy) with modern high performance processors is incompatible with the achievement of high levels of performance. Therefore, this form of shared memory computing is not discussed.

Experience has shown that the performance obtained from using *loop-level parallelism* is highly dependent on a number of factors. The four main factors are:

(1) The size and power of the computer system being used.

(2) The quality of the design and implementation of the computer system (both hardware, operating system, and programming environment) being used.

(3) The skill of the programmer who parallelizes and tunes the code, and the amount of time and effort this person is allowed to put into the project.

(4) The size and configuration of the data set (e.g., bigger is better, but usually only if one has a small number of large zones to work with, rather than a large number of small zones to work with).

Under optimal conditions, we have demonstrated both high levels of performance and moderate to high levels of speedup when using a 128-processor SGI Origin 2000. For reasons that are discussed later in this report, most of the other efforts/systems obtained lower levels of performance and/or speedup, although in many cases, the achieved level of performance was still quite useful and frequently cost effective.

This report highlights the difficulties associated with producing a highly scalable shared memory system, since without such systems, *loop-level parallelism* is not scalable. However, it is important to remember that many of the less scalable systems are exceedingly well-designed production quality systems. Furthermore, while the high-end vector-based systems have limited levels of scalability (16–32 processors), their delivered levels of performance can easily surpass that of a 128-processor SGI Origin 2000.

2. Operating System Constraints

There are several operating system constraints that limit the usefulness of a shared memory system and will therefore prevent one from achieving the desired level of performance when using *loop-level parallelism.* While some of these may seem to be trivial or obvious considerations, a large number of current and former systems would fail to meet our performance goals simply because of these limitations:

- Some systems have too limited an address range for the job's needs. This is partly based on the requirement to reserve a portion of the address range for each thread's private use. Furthermore, on many 32-bit systems, fully one-half of the address range is reserved for use solely by the kernel of the operating system. Finally, there has to be enough space left over to meet the job's other needs. In contrast, most codes implemented using MPI will not have this problem, since only a portion of the data needs to reside in each node's memory (remembering that with the message-passing model, each node executes a separate job with its own address range).

- If the operating system either implements a *master-slave* policy (as opposed to being an SMP), or fails to support a sufficiently fine-grained set of locks within the kernel, then it may have trouble efficiently supporting large numbers of processors. This can be especially troublesome to shared memory jobs, since they tend to hit synchronization barriers much more frequently than do message-passing jobs written with distributed memory systems in mind.

- The system must be run in a manner that allows a job using N processors to actually *own* those processors. Time sharing a processor is almost always counterproductive.

- The system must provide a programming environment that supports this form of parallelism.

In some cases, these constraints were by themselves sufficient to limit the usefulness of the early SMPs from SGI, SUN, and others. In fact, it wasn't until mid-1998 that the hardware and the operating system for the SGI Origin 2000 became sufficiently mature that 128-processor systems could be used in a production environment.

3. Hardware Constraints - The Obvious Issues

Hardware issues are a bit more complicated and are therefore considered in two separate groups. Some of the more obvious issues are:

- In general, if the system does not support more than N processors, most jobs have no hope of seeing speedups in excess of N.

- If the aggregate peak speed of a system does not exceed the estimated requirements to meet the user's needs (preferably by more than a factor of 2), the system will not be used.

- If the delivered level of performance per processor is a small percentage of peak (e.g., 1–2%), it is unlikely that there will be enough processors to meet the user's needs. Even if there are enough processors, it is difficult to see how the system will be cost effective. The net result is that the system is likely to be abandoned in short order (in fact, it might not even pass acceptance testing).

- The system must support *enough* physical memory. Additionally, it must provide the necessary hardware support for large address spaces.

• Only systems with a reputation for stability (or at least from a vendor with a reputation for building well-designed and stable supercomputers) are likely to be purchased for use as production systems.

• The system must be viewed as being affordable. This metric refers to the cost per **GFLOPS**, the cost per GB of memory, the maintenance costs, and, maybe most importantly, the cost for a minimally configured system. This final number was the forte of vendors of mini-supercomputers (e.g., Convex). When comparing vector-based systems to other classes of systems, it can be very difficult to properly evaluate these numbers (in part, because they depend on usage policies and what alternative systems are available). The net result is that frequently non-vector-based systems will appear to be more cost effective (at least to a degree) than they really are.

If a system fails to pass these tests, then it is unlikely to meet the user's needs.

4. Hardware Constraints - The Subtler Points

Some of the subtler points when it comes to the hardware are:

• The memory system needs to have a high bandwidth, low latency, and a cost that the customer is willing to pay. Unfortunately, it is difficult to create high-bandwidth, low-latency memory systems, and such systems are always expensive.

• A common solution to this issue is to use cache in an attempt to dramatically reduce the memory bandwidth requirements (e.g., by 99%). Unfortunately, this means that the code needs to be tuned in a manner that is cache friendly (vector-optimized code rarely starts this way, and most production scientific codes have been optimized for Cray vector processors). Even when the code has been retuned to be cache friendly (or at least one is willing and able to retune the code to be cache friendly), a bad cache design may make this job more difficult (or in some cases next to

impossible). In particular, this was our experience when Cray decided to design the Cray T3D and Cray T3E without any external cache [2].

- Many early attempts to build SMPs were based on designs using either *Buses* or *Cross Bar Switches*. System buses have the disadvantage of not being scalable. Therefore, as one adds processors to the system (or alternatively increases the performance of the processors), the system bus will rapidly become a key bottleneck to scalability. As a result, many of these designs supported limited numbers of processors and/or used multiple buses to support what was at best a moderate level of processing power. Vector-based SMPs have traditionally used cross bar switches to provide much higher levels of bandwidth. These switches were then coupled with very low-latency memory that was arranged in multiple banks to provide exceptional levels of memory bandwidth. Finally, the very design principles that vector processors are based on lend themselves to tolerating a fair amount of memory latency. Of course, all of that added expense to these systems, but it was necessary if the system was to perform well without a cache (something that is frequently of little value to a vector processor anyway).

- Attempts have been made to create large, scalable shared memory systems using **COMA** (KSR1) or **NUMA** (Convex Exemplar) architectures with long latencies and low bandwidths to the off-node memory. In general, these efforts have relied on some form of large DRAM-based cache to keep the requirement for off-node memory accesses to a minimum. Unfortunately, when working with 3D problems using *loop-level parallelism*, one is likely to find that some loops are parallelized in one direction, others in the second direction, and still others in the third direction. If there are dependencies present (as they were in key loops in the F3D code), it will be nearly impossible to avoid large-scale data motion (probably multiple times per time step per zone). This will make it difficult to show high levels of performance on these architectures.

- In the CRAFT model for the T3D, the required solution to the problem of off-node memory accesses is to *redistribute the data.* This is equivalent to using a matrix transpose to make all of the data local to the nodes where they will be processed. This option can also be used on COMA and

NUMA architectures. In fact, it is frequently used on any architecture with a memory hierarchy to improve the locality of reference. Unfortunately, it can be a very expensive operation to perform, and therefore its use is frequently minimized. While other approaches can also be used, their value can vary greatly from system to system.

- Experience with the SGI Origin 2000 indicates that it is possible to create an efficient NUMA architecture with an acceptable memory latency and bandwidth for off-node accesses.

- When using *loop-level parallelism* with 3D problems, one may encounter loops with dependencies in two out of the three directions. Those loops will in general exhibit limited amounts of parallelism (measured in the tens to the hundreds). This may be insufficient parallelism to take advantage of highly scalable architectures based on processors that are either weak or excessively inefficient.

- Since vector processing is a form of *instruction level parallelism*, attempts to use large numbers of vector processors in conjunction with *loop-level parallelism* may run into performance problems.

The important points here are:

- It has proven difficult to produce a low-cost, highly scalable memory system, without which one cannot hope to use large numbers of processors (or at least the customer base may be too small to be viable).

- If the systems require large numbers of processors and/or large vector lengths in order to achieve high levels of performance, then many candidates for parallelization using *loop-level parallelism* will simply lack the necessary parallelism.

5. The Programming Environment

There are at least four ways in which limitations in the programming environment may limit one's ability to demonstrate high levels of scalability when using *loop-level parallelism*:

(1) Some systems do not even have any compiler support for this type of parallelism (alternatively, some vendors who in theory provide support for this form of parallelism have chosen to de-emphasize this support).

(2) The compiler support might be so heavily based on automatic parallelization and/or on languages that are so rarely used at most sites that the support is of questionable value.

(3) On some systems, the implementation of *loop-level parallelism* introduces additional constraints on the job size, thereby decreasing the value of this support and/or limiting one's ability to show parallel speedup.

(4) If the normal usage of the system involves time-sharing the processors to any significant extent, then, at best, the observed level of parallel speedup will be highly variable, and, at worst, it can actually result in parallel slowdown.

Our experience with the CRAFT programming model on the CRAY T3D illustrates the case of excessive limitations inherent with the programming environment itself. CRAFT was the proposed worksharing programming model for the distributed-memory CRAY platforms. Like other *loop-level parallelism* approaches, it promised a relatively easy avenue of porting vector codes to the new generation of CRAY machines—in particular, the T3D. This F77 environment consisted of:

- CDIR$ SHARED compiler directives providing control over data distribution among the processing elements,

- CDIR$ DO SHARED directives providing control over low-level parallelism, and

- a set of subroutines and intrinsics to perform common data parallel operations such as reductions and scans.

Unfortunately, along with the ease of use and maintenance came important drawbacks. The parallel loop dimensions were restricted to powers of 2, leading to wasted memory and complicated array padding. Restrictions were placed on nesting of shared loops. Most importantly, the model did not allow almost any control over data access patterns inside of the shared loops, and the compiler itself had difficulty producing efficient code inside of those loops. The model also promoted the use of array syntax, which further reduced reuse of the cache. With the syntax different from the more widely supported **High Performance Fortran (HPF)** standard and uncompelling performance, CRAFT did not gain a lot of user support and was eventually placed by CRAY in a maintenance mode.

6. The F3D Code

Before continuing on with the results that were achieved by applying *loop-level parallelism* to the F3D code, there is a brief description of the code and what was done to it. F3D is an *implicit computational fluid dynamics* code (finite difference, evolved from the Beam-Warming [3] linearized block implicit technique) that was originally developed by J. L. Steger for the **NASA** Ames Research Center. Subsequent modifications to the code have been made by J. Sahu of the U.S. Army Research Laboratory [4].*

Originally, this code was written as an out-of-core solver (depending on the availability of a fast *solid-state disk* for good performance). The first step in any effort to parallelize this code was

* The original version of the code is called F3D. Since the efforts to parallelize the code resulted in large numbers of changes, the decision was made to rename the code. The parallelized vector version of the code is now referred to as F3DZONAL. The other two versions of the code contain additional extensions and are now collectively known as ZNSFLOW.

therefore to turn it into an in-core solver. However, this step created a few problems. On most Cray vector processors and smaller RISC-based shared memory systems (e.g., the SGI Power Challenge), this made the code a memory hog. This was not a big problem if one had a dedicated system, but in most other cases, it could complicate efforts to schedule these jobs. On distributed memory systems with small to moderate amounts of memory per node (e.g., the Cray T3D), the problem was even worse. In general, the memory requirements would set a lower bound on how many processors could be used. This can be an important limitation for a program with a limited amount of parallelism.

The original code consists of about 10,000 lines of Fortran 77, split into about 75 subroutines. Initially, the code was well vectorized but ran poorly on RISC-based processors and, on vector platforms, ran poorly with automatic compiler parallelization. The effort for the vector platforms consisted of additional efforts to improve the single processor performance of the code, along with the insertion of microtasking directives. The directives were inserted in the most time-consuming subroutines (about 15 of the 75). This involved declaring the variables as either shared or private and, in some cases, reordering the loops so as to maximize the number of parallel loops.

Additionally, some restructuring of the code was done to improve the parallelizability of the code (e.g., increasing the amount of work per processor per parallelization directive). This involved rearranging sections of code and, in a couple of cases, moving a code fragment from one subroutine to another.

The approach to parallelizing this code for RISC-based platforms was in theory quite similar. In practice, it was necessary to start this project with a concerted effort to improve the serial efficiency of the code on this new class of platforms. A number of techniques were used to dramatically reduce the number of cache misses that missed all the way back to the main memory and to reduce the Translation Lookaside Buffer (TLB). Furthermore, it was noted that since these processors are not vector processors, it was in most cases possible to remove optimizations aimed at improving the vectorizability of the code. In some cases, this allowed us to reduce the cache and TLB miss rates

(sometimes quite dramatically). In other cases, it resulted in modest reductions in the number of floating point operations per time step per grid point. One major benefit of the observation that the code no longer needed to be vectorizable was that one could now use 100% of the available parallelism for parallelizing the program. This observation greatly increased the level of performance that we could achieve on the SGI Origin 2000. Furthermore, it was critical to the porting of the code to the Cray T3D.

Finally, as we have seen in the previous section, while it was expected that the version of the code running on the Cray T3D would use compiler directives similar to those used on the vector-based platforms from Cray and the RISC-based platforms from SGI and Convex, things did not work out that way. That one of us was able to implement *loop-level parallelism* using message-passing code on the Cray T3D and the Cray T3E demonstrates the value of *loop-level parallelism.* However, the difficulty of using this approach, and the need to use it in the first place, substantially places in doubt the validity of Cray Research's frequent claims that the T3D and T3E are shared memory systems.

7. Results

In addition to the work that we did with F3D, we are aware of work done with a number of other programs at NASA Ames Research Center and at ARL. A summary of these results are:

- *Automatic parallelizing compilers* rarely show much speedup on their own. In fact, it is not uncommon to observe parallel slowdown when going from one to two processors [5].

- Some jobs that make heavy use of Fortran 90 Array syntax show parallel speedup when using fully automatic parallelization. However, since many serial compilers produce an excessively high cache miss rate when compiling code written in this manner, the performance of parallel code written in this manner should be considered suspect.

- As indicated in section 5, attempts to use the CRAFT model with the T3D computer produced exceptionally poor levels of performance (3% of peak). The poor performance was seen across the board, from parts of F3D that necessitated complex data motion (block tridiagonal solver) to parts consisting of *embarrassingly* parallel operations (formation of residuals). A later implementation of the F3D on CRAY **MPP**s took advantage of the message-passing (still using the concept of *loop-level parallelism*) and performed at a more acceptable 20% of peak. The data motion necessary in the block tridiagonal solver and other sections of the code are now performed explicitly, providing more opportunities to store intermediate results and to aggregate sends and receives, resulting in much reduced communication overhead. The remaining communication-free portions of the code bear a strong resemblance to scalar RISC counterparts, and optimization lessons learned there can be applied. The message-passing version is based on the MPI standard and can also take advantage of the low-latency SHMEM communication library, available on CRAY **MPP**s and SGI SMPs. While the resulting code is portable across a wide range of machines (CRAY, SGI, IBM SP, and Sun HPC), a major effort was needed to insert explicit communication mechanisms, making this approach impractical in some production environments.

- The combination of using automatic parallelization with compiler directives (e.g., C$doacross) will in general produce better levels of performance than when using automatic parallelization on its own. However, the actual levels of speedup achieved are highly dependent on the level of manual tuning and do not always show much benefit past 16 processors (results from runs performed on SGI Origin 2000s) [5, 6].

- On a J932/16, the original code ran with a speedup of 3.19 when using 15 CPUs and automatic parallelization. The optimized version of the code with its microtasking directives ran 12.22 times faster on 15 CPUs than it did on one CPU. So the optimization effort resulted in the code running four times faster than the original code with automatic parallelization.

- On a Cray T90, the optimized code ran 1,800 time steps of a 1 million grid point test case in 12,720 s on one CPU. When using four CPUs, the same test case ran in 4,656 s (the system was not

dedicated). This is a speedup of 2.7. Presumably, larger problems would perform even better; however, memory constraints limit the range of problem sizes that can be run within core solvers on the Cray T90.

- Attempts to use *loop-level parallelism* primarily/solely based on compiler directives (with heavy emphasis on code tuning for both serial and parallel performance) have consistently shown good results on SGI Origin 2000s. Depending on the choice of problem, problem size, and system size, at least three separate groups of researchers have reported speedups ranging from 10 to 80. Efforts involving systems from Convex as well as older systems from SGI usually were not as successful [5, 7–9].

- James Taft of **NASA** Ames Research Center/MRJ Technology Solutions Inc. has developed a new approach for using multiple levels of *loop-level parallelism* combined with extra code to perform load balancing. He has been reporting very favorable results when using this approach on large SGI Origin 2000s.

8. Conclusion

Achieving highly scalable results when using *loop-level parallelism* is possible. Having said this, we now note that when starting a new project, one should not assume that one will automatically achieve highly scalable results. Achieving this goal requires a well-designed and implemented computer system (including the operating system and programming environment). Additionally, it requires a significant commitment on the part of the programmer to *do it right.* By this, we mean do the serial and parallel tuning and do not expect the compiler to do more than a limited amount of the work for the programmer.

INTENTIONALLY LEFT BLANK.

9. References

1. Dowd, K., and C. Severance. *High Performance Computing: RISC Architectures, Optimization & Benchmarks 2nd Edition.* Sebastopol, CA: O'Reily & Associates, Inc., 1998.

2. O'Neal, D., and J. Urbanic. "On Performance and Efficiency: Cray Architectures." http://www.psc.edu/~oneal/eff/eff.html, Parallel Applications Group, Pittsburgh Supercomputing Center, August 1997.

3. Pulliam, T., and J. Steger. "On Implicit Finite-Difference Simulations of Three-Dimensional Flow." *AIAA Journal*, vol. 18, pp. 159–167, 1992.

4. Sahu, J., and J. L. Steger. "Numerical Simulation of Transonic Flows." *International Journal for Numerical Methods in Fluids*, vol. 10, no. 8, 1990, pp. 855–873.

5. Frumking, M., M. Hribar, H. Jin, A. Waheed, and J. Yan. "A Comparison of Automatic Parallelization Tools/Compilers on the SGI Origin 2000." Proceedings for SC98, ACM, November 1998.

6. Hisley, D. M., G. Agrawal, and L. Pollock. "Performance Studies of the Parallelization of a CFD Solver on the Origin 2000." Proceedings for the 21st Army Science Conference, June 1998.

7. Pressel, D. M., W. B. Sturek, J. Sahu, and K. R. Heavey. "How Moderate-Sized RISC-Based SMPs Can Outperform Much Larger Distributed Memory MPPs." To be published in the conference proceedings for the 1999 Advanced Simulation Technologies Conference, San Diego, CA, sponsored by The Society for Computer Simulation International (SCS), 11–15 April 1999.

8. Saini, S. (editor). NAS Parallel Benchmarks. NPB 1 Data, Electronically published at http://science.nas.nasa.gov/Software/NPB/NPB1Results/index.html, 17 November 1996.

9. Pressel, D. M. "Results From the Porting of the Computational Fluid Dynamics Code F3D to the Convex Exemplar CSPP-1000 and SPP-1600." ARL-TR-1923, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, March 1999.

INTENTIONALLY LEFT BLANK.

Glossary

CFD	Computational Fluid Dynamics
CISC	Complicated Instruction Set Computer, an approach to processor design that assumes that the best way to get good performance out of a system is to provide instructions that are designed to implement key constructs (e.g., loops) from high level languages.
COMA	Cache Only Memory Architecture
GB	Billion Bytes
GFLOPS	Billion floating point operations per second
High Level Languages	Computer languages that are designed to be relatively easy for the programmer to read and write. Examples of this type of language are Fortran, Cobol, C, etc.
HPF	High Performance Fortran
Low Level Languages	Computer languages that are designed to reflect the actual instruction set of a particular computer. In general, the lowest level language is known as Machine Code. Just slightly above machine code is a family of languages collectively known as Assembly Code.
MPP	Massively Parallel Processor
NASA	National Aeronautics and Space Administration
NUMA	Non Uniform Memory Access
RISC	Reduced Instruction Set Computer, an approach to processor design that argues that the best way to get good performance out of a system is to eliminate the Micro Code that CISC systems use to implement most of their instructions. Instead, all of the instructions will be directly implemented in hardware. This places obvious limits on the complexity of the instruction set, which is why the complexity had to be *reduced.*
SMP	Symmetric Multiprocessor

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	DEFENSE TECHNICAL INFORMATION CENTER DTIC DDA 8725 JOHN J KINGMAN RD STE 0944 FT BELVOIR VA 22060-6218
1	HQDA DAMO FDQ D SCHMIDT 400 ARMY PENTAGON WASHINGTON DC 20310-0460
1	OSD OUSD(A&T)/ODDDR&E(R) R J TREW THE PENTAGON WASHINGTON DC 20301-7100
1	DPTY CG FOR RDA US ARMY MATERIEL CMD AMCRDA 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	INST FOR ADVNCD TCHNLGY THE UNIV OF TEXAS AT AUSTIN PO BOX 202797 AUSTIN TX 78720-2797
1	DARPA B KASPAR 3701 N FAIRFAX DR ARLINGTON VA 22203-1714
1	NAVAL SURFACE WARFARE CTR CODE B07 J PENNELLA 17320 DAHLGREN RD BLDG 1470 RM 1101 DAHLGREN VA 22448-5100
1	US MILITARY ACADEMY MATH SCI CTR OF EXCELLENCE DEPT OF MATHEMATICAL SCI MADN MATH THAYER HALL WEST POINT NY 10996-1786

NO. OF COPIES	ORGANIZATION
1	DIRECTOR US ARMY RESEARCH LAB AMSRL DD 2800 POWDER MILL RD ADELPHI MD 20783-1197
1	DIRECTOR US ARMY RESEARCH LAB AMSRL CS AS (RECORDS MGMT) 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LAB AMSRL CI LL 2800 POWDER MILL RD ADELPHI MD 20783-1145
	ABERDEEN PROVING GROUND
4	DIR USARL AMSRL CI LP (BLDG 305)

NO. OF COPIES	ORGANIZATION
1	PM CHSSI JOHN GROSH SUITE 510 1010 N GLEBE ROAD ARLINGTON VA 22201
1	RICE UNIVERSITY MCHNCL ENGRNG AND MTRLS SCI MAREK BEHR MS 321 6100 MAIN STREET HOUSTIN TX 77005
1	COMMANDER CODE C2892 CLINT HOUSH 1 ADMINISTRATION CIR CHINA LAKE CA 93555
2	WL FIMC STEPHEN SCHERR BILL STRANG BLDG 450 2645 FIFTH ST SUITE 7 WPAFB OH 45433-7913
1	NSWC A B WARDLAW CODE B44 SILVER SPRING MD 20903-5640
1	NAVAL RSRCH LAB CODE 6400 JAY BORIS 4555 OVERLOOK AVE SW WASHINGTON DC 20375-5344
1	NAVAL RSRCH LAB CODE 6410 RAVI RAMAMURTI WASHINGTON DC 20375-5344
1	ARMY AEROFLIGHT DYNAMICS DIRECTORATE ROBERT MEAKIN MS 258 1 MOFFETT FIELD CA 94035-1000
1	NAVAL RSRCH LAB CODE 7320 J W MCCAFFREY JR HEAD OCEAN DYNAMICS AND PREDICTION BRANCH STENNIS SPACE CENTER MS 39529

NO. OF COPIES	ORGANIZATION
1	NAVAL RSRCH LAB GEORGE HEBURN RSRCH OCEANOGRAPHER CNMOC BLDG 1020 RM 178 STENNIS SPACE CENTER MS 39529
1	US AIR FORCE WRIGHT LAB WL FIM JOSEPH J S SHANG 2645 FIFTH STREET STE 6 WPAFB OH 45433-7912
1	USAF PHILIPS LAB OLAC PL RKFE CPT SCOTT G WIERSCHKE 10 EAST SATURN BLVD EDWARDS AFB CA 93524-7680
1	USAE WATERWAYS EXPERIMENT STATION CEWES HV C JEFFREY P HOLLAND 3909 HALLS FERRY ROAD VICKSBURG MS 39180-6199
1	US ARMY CECOM RD&E CTR AMSEL RD C2 BARRY S PERLMAN FT MONMOUTH NJ 07703
1	SPAWARSYSCEN (D4402) ROBERT A WASILAUSKY BLDG 33 RM 0071A 53560 HULL ST SAN DIEGO CA 92152-5001
1	US AIR FORCE RESEARCH LAB INFORMATION DIRECTORATE RICHARD W LINDERMAN 26 ELECTRONIC PARKWAY ROME NY 13441-4514
1	US AIR FORCE RESEARCH LAB PROPULSION DIRECTORATE LESLIE PERKINS 5 POLLUX DR EDWARDS AFB CA 93524-7048
1	AIR FORCE RESEARCH LAB/DEHE ROBERT PETERKIN 3550 ABERDEEN AVE SE KIRTLAND AFB NM 87117-5776

NO. OF
COPIES ORGANIZATION

1 SPACE & NAVAL WARFARE SYS CTR
CODE D7305 KEITH BROMLEY
BLDG 606 RM 325
53140 SYSTEMS ST
SAN DIEGO CA 92152-5001

1 UNVRSTY OF MINNESOTA
DEPT OF ASTRONOMY
PROF P WOODWARD
356 PHYSICS BLDG
116 CHURCH STREET SE
MINNEAPOLIS MN 55455

1 RICE UNIVERSITY
MCHNCL ENGRNG AND MTRLS SCI
TAYFUN TEZDUYAR DEPT CHRMN
MS 321 6100 MAIN ST
HOUSTON TX 77005

1 DIRECTOR
ARMY HIGH PERFORMANCE
COMPUTING RSRCH CTR
BARBARA BRYAN
1200 WASHINGTON AVE
SOUTH MINNEAPOLIS MN 55415

1 DIRECTOR
ARMY HIGH PERFORMANCE
COMPUTING RSRCH CTR
GRAHAM V CANDLER
1200 WASHINGTON AVE
SOUTH MINNEAPOLIS MN 55415

1 NAVAL CMND CONTROL AND
OCEAN SURVEILLANCE CTR
L PARNELL HPC CRDNTR & DIR
NCCOSC RDTE DIV D3603
49590 LASSING ROAD
SAN DIEGO CA 92152-6148

NO. OF
COPIES ORGANIZATION

ABERDEEN PROVING GROUND

15 DIR USARL
AMSRL CI
N RADHAKRISHNAN
AMSRL CI H
C NIETUBICZ
AMSRL CI HA
W STUREK
A MARK
R NAMBURU
AMSRL CI HC
D PRESSEL
D HISLEY
C ZOLTANI
A PRESSLEY
T KENDALL
P DYKSTRA
AMSRL WM BC
H EDGE
J SAHU
K HEAVEY
P WEINACHT

INTENTIONALLY LEFT BLANK.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 2000	Final, Jan 96 - Dec 96

4. TITLE AND SUBTITLE

The True Limitations of Shared Memory Programming

5. FUNDING NUMBERS

9UHMCL

6. AUTHOR(S)

D. M. Pressel, M. Behr,* and S. Thompson**

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Research Laboratory
ATTN: AMSRL-CI-HC
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER

ARL-TR-2147

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10.SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

*University of Minnesota, 1100 Washington Avenue, Minneapolis, MN 55415
**Raytheon Systems Company, 939-I Beards Hill Road, Suite 191, Aberdeen, MD 21001

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Shared memory parallel computers have the reputation for being the easiest type of parallel computers to program. At the same time, they are frequently regarded as being the least scalable type of parallel computer. In particular, shared memory parallel computers are frequently programmed using a form of loop-level parallelism (usually based on some combination of compiler directives and automatic parallelization). However, in discussing this form of parallelism, the experts in the field routinely say that it will not scale past 4–16 processors (the number varies among experts). This report investigates what the true limitations are to this type of parallel programming. The discussions are largely based on the experiences that the authors had in porting the Implicit Computational Fluid Dynamics Code (F3D) to numerous shared memory systems from SGI, Cray, and Convex.

14. SUBJECT TERMS

supercomputer, high performance computing, parallel programming, symmetric multiprocessors

15. NUMBER OF PAGES

22

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number/Author ARL-TR-2147 (Pressel) Date of Report January 2000

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name E-mail Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)

(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001,APG,MD

POSTAGE WILL BE PAID BY ADDRESSEE

DIRECTOR
US ARMY RESEARCH LABORATORY
ATTN AMSRL CI HC
ABERDEEN PROVING GROUND MD 21005-5067